IN THE CLAIMS:

The status and content of each claim follows:

1. (currently amended) A method of optimizing the performance of an interpreter based runtime system, the runtime system including a virtual machine, the virtual machine adapted to run an application in the context of the runtime environment, the method comprising:

augmenting a bytecode set of the virtual machine with semantically enriched opcodes, thereby constituting an application domain-specific virtual machine;

optimizing the virtual machine based on semantics of the application to be run on the virtual machine, with at least a portion of the semantically enriched opcodes being specific to the application;

performing a quantitative trade-off between execution time and memory space to determine effective semantically enriched opcodes and encoding the semantically enriched opcodes into interpreter action codes based upon the trade-off;

analyzing frequently executed bytecodes and encoding the semantically enriched opcodes into interpreter action codes of the instruction set of the virtual machine to efficiently decode the ~~decode~~ frequently executed bytecodes;

optimizing the translation by the interpreter action codes of the semantically enriched opcodes according to a system state, said system state being represented by at least one symbolic variable; and

statically embedding the semantically enriched opcode to optimize execution of the interpreter-based runtime system.

2-20. (cancelled)

21. (previously presented) The method of claim 1, further comprising analyzing an application code using static optimization, the static optimization comprising parsing the application code to identify at least one repeated sequence of bytecodes and replace the at least one repeated sequence of bytecodes with a semantically enriched opcode.

22. (previously presented) The method of claim 1, further comprising analyzing an application code using dynamic optimization, the dynamic optimization comprising:

analyzing temporal behavior of the application code during execution; and

identifying and replacing at least one repeated computational sequence in a bytecode stream with a semantically enriched opcode.

23. (previously presented) The method of claim 1, further comprising:

discovering at least one repetitive computational sequence used in a symbolic state; and

generating a semantically enriched opcode corresponding to the at least one repetitive computational sequence used in the symbolic state.

24. (previously presented) The method of claim 23, wherein the symbolic state comprises at least one of: control flow, data flow, entry points, and operational arguments.

25. (previously presented) The method of claim 1, further comprising optimizing native code output by the virtual machine using a global optimizing compiler.

26. (previously presented) The method of claim 1, further comprising optimizing the virtual machine by offline compilation of the virtual machine by a host device.

27. (previously presented) The method of claim 1, further comprising optimizing the virtual machine by online modification of a generic virtual machine by inserting a stub, the stub automatically loading a semantically enriched opcode when the virtual machine encounters an identified code segment with a bytestream.

28. (previously presented) The method of claim 1, further comprising offline embedding of the semantically enriched opcodes in an application by substituting a semantically enriched opcode for a corresponding code segment.

29. (previously presented) The method of claim 1, further comprising online embedding of semantically enriched opcodes by a class loader, said class loader substituting a semantically enriched opcode for a corresponding code segment.

30. (currently amended) A system for optimizing performance of an interpreter based runtime system, the runtime system including a virtual machine, the system comprising:

application code;

an embedded processor;

a virtual machine configured to translate said application code into native machine code compatible with said embedded processor;

a detection module, said detection module being configured to analyze said application code to identify code segments that could be efficiently represented as semantically enriched opcodes, at least a portion of said semantically enriched opcodes being specific to said application;

an embedding module, said embedding module being configured to embed said semantically enriched opcodes in said application;

a code generation module, said code generation module being configured to generate optimized action code for translating said semantically enriched opcodes according to symbolic states, each of said symbolic states being represented by at least one symbolic variable; and

a build module configured create an application domain-specific virtual machine by incorporating said optimized action code and a bytecode set comprising said semantically enriched opcodes into said virtual machine.

31. (previously presented) The system of claim 30, wherein said detection module is configured to analyze said application code using static optimization, said static optimization comprising parsing said application code to identify at least one repeated sequence of bytecodes and replace said at least one repeated sequence of bytecodes with a semantically enriched opcode.

32. (currently amended) The system of claim 31, wherein said detection module is further configured to analyze said application code using dynamic optimization, said dynamic optimization comprising:

analyzing temporal behavior of the application code during execution; and

identifying and replacing at least one repeated computational sequence in a bytecode stream with [[an]] a semantically enriched opcode.

33. (previously presented) The system of claim 30, wherein said generation module is further configured to:

discover at least one repetitive computational sequence used in a said symbolic state; and

generate a semantically enriched opcode corresponding to the at least one repetitive computational sequence used within said symbolic state.

34. (previously presented) The system of claim 33, wherein said symbolic state comprises at least one of: control flow, data flow, entry points, and operational arguments.

35. (previously presented) The system of claim 30, further comprising a global optimizer compiler configured to optimizing native code output by said virtual machine.

36. (previously presented) The system of claim 30, wherein said virtual machine is compiled offline by a host device.

37. (previously presented) The system of claim 30, wherein said virtual machine is modified online by inserting a stub, said stub automatically loading a semantically enriched opcode when said virtual machine encounters an identified code segment with a bytestream.

38. (previously presented) The system of claim 30, wherein said application code is modified offline by substituting a semantically enriched opcode for a corresponding code segment contained with said application code.

39. (previously presented) The system of claim 30, wherein a class loader embeds said semantically enriched opcodes online, said class loader substituting a semantically enriched opcodes for a corresponding code segment.